Enterprise Programming
6G6Z1103

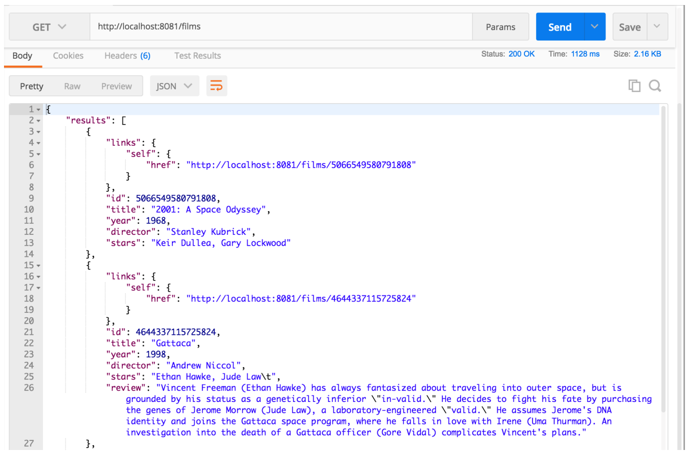
Joshua Michael Ephraim Bridge
14032908
joshua.m.bridge@stu.mmu.ac.uk

January 12, 2018

Contents

1 Completed Tasks

1.1 HTTP Web Service

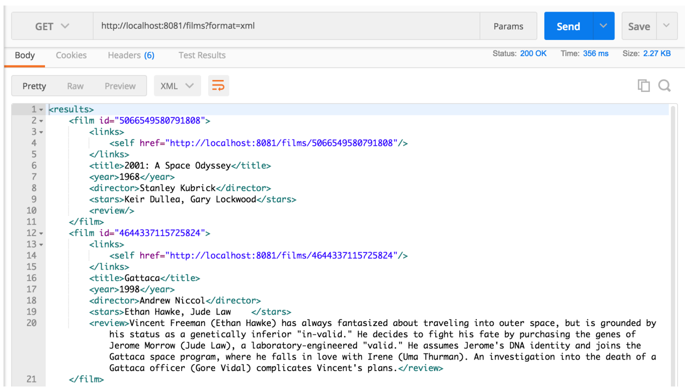


```
{
    "results": [
        {
            "links": {
                "self": {
                    "href": "http://localhost:8081/films/5066549580791808"
                }
            },
            "id": 5066549580791808,
            "title": "2001: A Space Odyssey",
            "year": 1968,
            "director": "Stanley Kubrick",
            "stars": "Keir Dullea, Gary Lockwood"
        },
        {
            "links": {
                "self": {
                    "href": "http://localhost:8081/films/4644337115725824"
                }
            },
            "id": 4644337115725824,
            "title": "Gattaca",
            "year": 1998,
            "director": "Andrew Niccol",
            "stars": "Ethan Hawke, Jude Law\t",
            "review": "Vincent Freeman (Ethan Hawke) has always fantasized about traveling into outer space, but is
                grounded by his status as a genetically inferior \"in-valid.\" He decides to fight his fate by purchasing
                the genes of Jerome Morrow (Jude Law), a laboratory-engineered \"valid.\" He assumes Jerome's DNA
                identity and joins the Gattaca space program, where he falls in love with Irene (Uma Thurman). An
                investigation into the death of a Gattaca officer (Gore Vidal) complicates Vincent's plans."
        },
```

Figure 1: Http web service call

1.2 Data formats

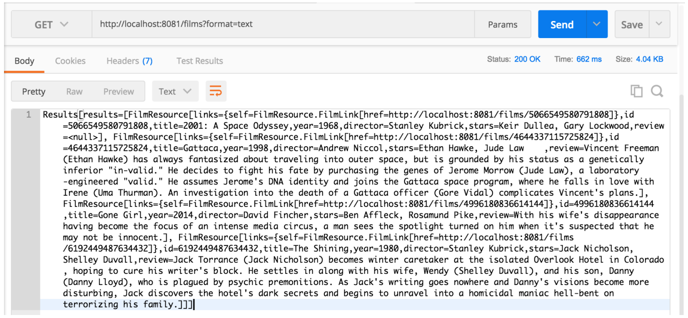
For JSON see figure 1.



```
http://localhost:8081/films?format=xml
```

```
<results>
    <film id="5066549580791808">
        <links>
            <self href="http://localhost:8081/films/5066549580791808"/>
        </links>
        <title>2001: A Space Odyssey</title>
        <year>1968</year>
        <director>Stanley Kubrick</director>
        <stars>Keir Dullea, Gary Lockwood</stars>
        <review/>
    </film>
    <film id="4644337115725824">
        <links>
            <self href="http://localhost:8081/films/4644337115725824"/>
        </links>
        <title>Gattaca</title>
        <year>1998</year>
        <director>Andrew Niccol</director>
        <stars>Ethan Hawke, Jude Law    </stars>
        <review>Vincent Freeman (Ethan Hawke) has always fantasized about traveling into outer space, but is grounded by
            his status as a genetically inferior "in-valid." He decides to fight his fate by purchasing the genes of
            Jerome Morrow (Jude Law), a laboratory-engineered "valid." He assumes Jerome's DNA identity and joins the
            Gattaca space program, where he falls in love with Irene (Uma Thurman). An investigation into the death of a
            Gattaca officer (Gore Vidal) complicates Vincent's plans.</review>
    </film>
```

Figure 2: XML web service call

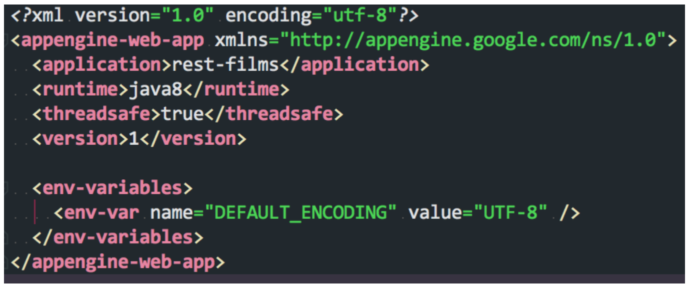


```
http://localhost:8081/films?format=text
```

```
Results[results=[FilmResource[links={self=FilmResource.FilmLink[href=http://localhost:8081/films/5066549580791808]},id
    =5066549580791808,title=2001: A Space Odyssey,year=1968,director=Stanley Kubrick,stars=Keir Dullea, Gary Lockwood,review
    =<null>], FilmResource[links={self=FilmResource.FilmLink[href=http://localhost:8081/films/4644337115725824]},id
    =4644337115725824,title=Gattaca,year=1998,director=Andrew Niccol,stars=Ethan Hawke, Jude Law    ,review=Vincent Freeman
    (Ethan Hawke) has always fantasized about traveling into outer space, but is grounded by his status as a genetically
    inferior "in-valid." He decides to fight his fate by purchasing the genes of Jerome Morrow (Jude Law), a laboratory
    -engineered "valid." He assumes Jerome's DNA identity and joins the Gattaca space program, where he falls in love with
    Irene (Uma Thurman). An investigation into the death of a Gattaca officer (Gore Vidal) complicates Vincent's plans.],
    FilmResource[links={self=FilmResource.FilmLink[href=http://localhost:8081/films/4996180836614144]},id=4996180836614144
    ,title=Gone Girl,year=2014,director=David Fincher,stars=Ben Affleck, Rosamund Pike,review=With his wife's disappearance
    having become the focus of an intense media circus, a man sees the spotlight turned on him when it's suspected that he
    may not be innocent.], FilmResource[links={self=FilmResource.FilmLink[href=http://localhost:8081/films
    /6192449487634432]},id=6192449487634432,title=The Shining,year=1980,director=Stanley Kubrick,stars=Jack Nicholson,
    Shelley Duvall,review=Jack Torrance (Jack Nicholson) becomes winter caretaker at the isolated Overlook Hotel in Colorado
    , hoping to cure his writer's block. He settles in along with his wife, Wendy (Shelley Duvall), and his son, Danny
    (Danny Lloyd), who is plagued by psychic premonitions. As Jack's writing goes nowhere and Danny's visions become more
    disturbing, Jack discovers the hotel's dark secrets and begins to unravel into a homicidal maniac hell-bent on
    terrorizing his family.]]]
```

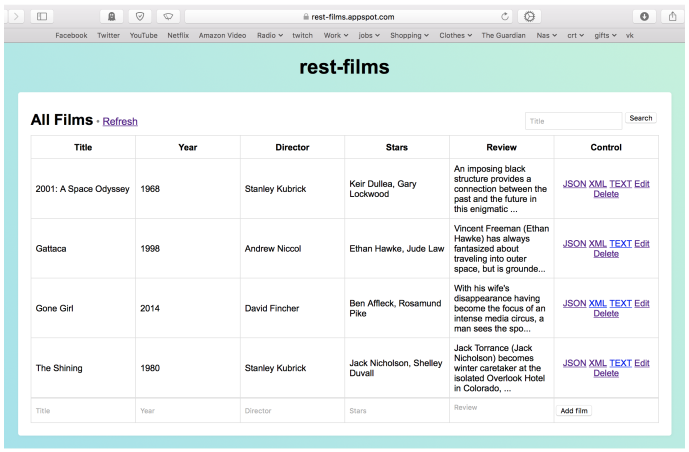
Figure 3: TEXT web service call

1.3 Appengine



```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>rest-films</application>
  <runtime>java8</runtime>
  <threadsafe>true</threadsafe>
  <version>1</version>

  <env-variables>
    <env-var name="DEFAULT_ENCODING" value="UTF-8" />
  </env-variables>
</appengine-web-app>
```

Figure 4: Appengine config


rest-films.appspot.com
Facebook Twitter YouTube Netflix Amazon Video Radio twitch Work jobs Shopping Clothes The Guardian Nas crt gifts vk

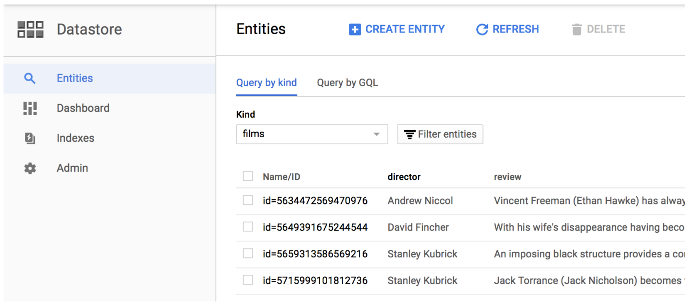
rest-films

All Films · Refresh

Title Search

Title	Year	Director	Stars	Review	Control
2001: A Space Odyssey	1968	Stanley Kubrick	Keir Dullea, Gary Lockwood	An imposing black structure provides a connection between the past and the future in this enigmatic ...	JSON XML TEXT Edit Delete
Gattaca	1998	Andrew Niccol	Ethan Hawke, Jude Law	Vincent Freeman (Ethan Hawke) has always fantasized about traveling into outer space, but is grounde...	JSON XML TEXT Edit Delete
Gone Girl	2014	David Fincher	Ben Affleck, Rosamund Pike	With his wife's disappearance having become the focus of an intense media circus, a man sees the spo...	JSON XML TEXT Edit Delete
The Shining	1980	Stanley Kubrick	Jack Nicholson, Shelley Duvall	Jack Torrance (Jack Nicholson) becomes winter caretaker at the isolated Overlook Hotel in Colorado, ...	JSON XML TEXT Edit Delete
Title	Year	Director	Stars	Review	Add film

Figure 5: Appengine Deployment (`http://rest-films.appspot.com`)



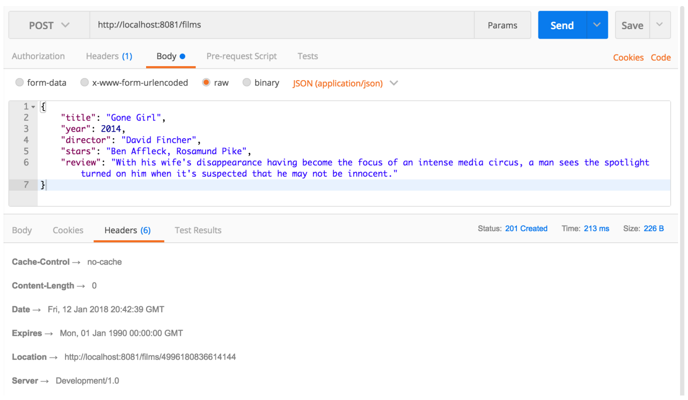
Name/ID	director	review
id=5634472569470976	Andrew Niccol	Vincent Freeman (Ethan Hawke) has alway
id=5649391675244544	David Fincher	With his wife's disappearance having beco
id=5659313586569216	Stanley Kubrick	An imposing black structure provides a co
id=5715999101812736	Stanley Kubrick	Jack Torrance (Jack Nicholson) becomes

Figure 6: Datastore entries

1.4 WSDL

I could not find a way to convert my java classes to WSDL, I tried it in multiple applications and it never worked. Every example given in the lab exercises was for a SOAP webservice so as my project is mainly a REST service, a WSDL does not seem to apply to it.

1.5 REST



POST http://localhost:8081/films

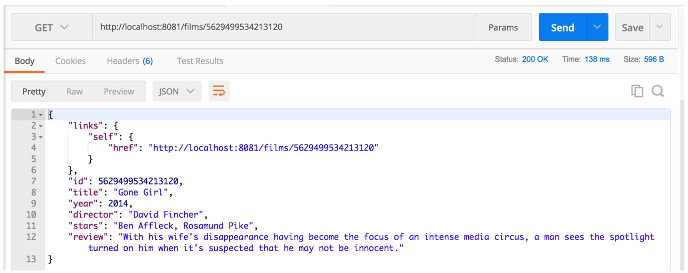
JSON (application/json)

```
{
    "title": "Gone Girl",
    "year": 2014,
    "director": "David Fincher",
    "stars": "Ben Affleck, Rosamund Pike",
    "review": "With his wife's disappearance having become the focus of an intense media circus, a man sees the spotlight
        turned on him when it's suspected that he may not be innocent."
}
```

Status: 201 Created Time: 213 ms Size: 226 B

```
Cache-Control → no-cache
Content-Length → 0
Date → Fri, 12 Jan 2018 20:42:39 GMT
Expires → Mon, 01 Jan 1990 00:00:00 GMT
Location → http://localhost:8081/films/4996180836614144
Server → Development/1.0
```

Figure 7: Film resource create (201 Response - with resource location)

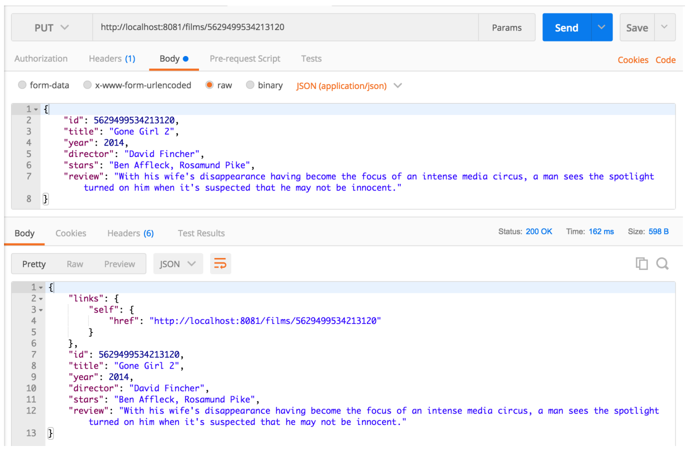


GET http://localhost:8081/films/5629499534213120

Status: 200 OK Time: 138 ms Size: 596 B

```
{
    "links": {
        "self": {
            "href": "http://localhost:8081/films/5629499534213120"
        }
    },
    "id": 5629499534213120,
    "title": "Gone Girl",
    "year": 2014,
    "director": "David Fincher",
    "stars": "Ben Affleck, Rosamund Pike",
    "review": "With his wife's disappearance having become the focus of an intense media circus, a man sees the spotlight
        turned on him when it's suspected that he may not be innocent."
}
```

Figure 8: Film resource response



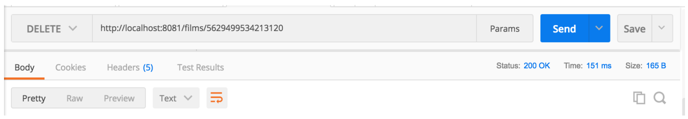
```
http://localhost:8081/films/5629499534213120
```

```
{
    "id": 5629499534213120,
    "title": "Gone Girl 2",
    "year": 2014,
    "director": "David Fincher",
    "stars": "Ben Affleck, Rosamund Pike",
    "review": "With his wife's disappearance having become the focus of an intense media circus, a man sees the spotlight
       turned on him when it's suspected that he may not be innocent."
}
```

Status: 200 OK Time: 162 ms Size: 598 B

```
{
    "links": {
        "self": {
            "href": "http://localhost:8081/films/5629499534213120"
        }
    },
    "id": 5629499534213120,
    "title": "Gone Girl 2",
    "year": 2014,
    "director": "David Fincher",
    "stars": "Ben Affleck, Rosamund Pike",
    "review": "With his wife's disappearance having become the focus of an intense media circus, a man sees the spotlight
       turned on him when it's suspected that he may not be innocent."
}
```

Figure 9: Film resource update (200 Response)



```
http://localhost:8081/films/5629499534213120
```

Status: 200 OK Time: 151 ms Size: 165 B

Figure 10: Film resource deletion (200 Response)

1.6 AJAX Front-end

See figure 5.

2 Design Patterns

2.1 Model-view-controller (MVC)

The Model-view-controller (Krasner et al. 1988) design pattern is one of the most commonly used methods for web development. It has become so essential due to its simplicity, and how it easily separates concerns from different sections of a program. For example a controller class should not have to worry about how to retrieve data from a table - as that is not part of its purpose and will only make the class harder to comprehend and work with. With clear separation of concerns, development in any language or platform becomes a lot simpler. I have used the MVC pattern in my assignment as it was the obvious choice for a simple requirement of displaying film data.

However, I did not use the ‘view’ portion of the pattern as the main component was an API returning JSON, XML and TEXT data - and instead of rendering JSP pages I used the Controller itself to render the responses. To do this I used a framework called Spring MVC (see section 4.2).

2.2 Data Access Object (DAO)

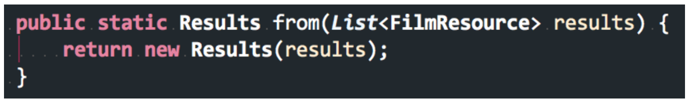
Data access objects are a very fundamental design pattern which provides access to external services such as database connections or a connection to an external API. In my application I have used the DAO pattern to connect to Google Datastore (a type of NoSql database).

2.3 Singleton

The Singleton design pattern is used by default in a Spring bean. For example, a class annotated with ‘@Component’ is instantiated and held in the Spring context as a single instance of that class. Whenever you want to use this object you can simply wire it into other fields using the ‘@Autowired’ annotation as shown in figure 15. This pattern removes a lot of the hassle for managing different instances of classes, especially as in this situation where multiple instances of controllers and DAO’s are not needed so was very effective in my application.

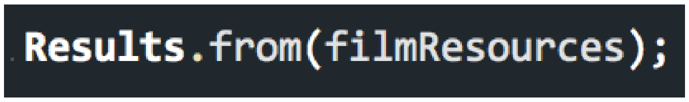
2.4 Static Factory

I have used a static factory in my application (as seen in figure 11). Static factories serve to make code more readable when instantiating objects. This can be seen in figure 12 where a Results object is created, and the code actually reads like a sentence - making it much easier to understand what is happening without spending too much time trying to read through it.



```
public static Results from(List<FilmResource> results) {
    return new Results(results);
}
```

Figure 11: Static factory method



```
Results.from(filmResources);
```

Figure 12: Static factory method usage

3 Refactoring

Refactoring is an essential part of developing - especially in a team - in order to make sure your code is easily readable and maintainable. This is especially necessary in a team as other people will often be looking at your code, therefore if it is too difficult to understand then it wastes developers time and money trying to understand it rather than extending it.

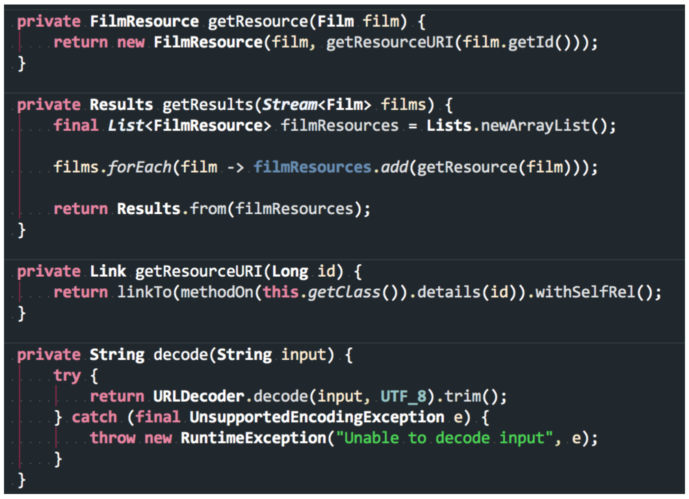
One of the best ways to ensure readable code is to make sure methods and classes only have a single purpose. As shown in figure 13, my FilmController class has a set of private utility methods which handle common tasks such as converting a Film object into a web resource (FilmResource). These methods are easy to understand as they only perform a single purpose, and the naming of the methods should make it understandable enough that even if the contents of the method are not understandable, then it's purpose still is clear and the code can still easily be maintained.

An extension of this is to try and stick to 3 line functions (at least 3 meaningful lines). This is a useful technique as it helps maintain low complexity in any code and ensures an application is split up into meaningful section, which also can be seen in figure 13.

In figure 3 you can see the getter/setter methods on the Film class, and I have removed the setter methods for everything except from setID. This is due to the fact that originally the fields were immutable, however as explained in 4.4 I had to remove this for multiple reasons. I decided not to add setters for the rest as there was no code that needed them so it would have been pointless, and would have just cluttered up the code.

4 Libraries (& Tools)

In this section I will talk a little about each of the main third-party libraries I used, why I chose to use them, and how it helped me solve a problem I was facing - or how it reduced reliance on custom code. Reducing as much



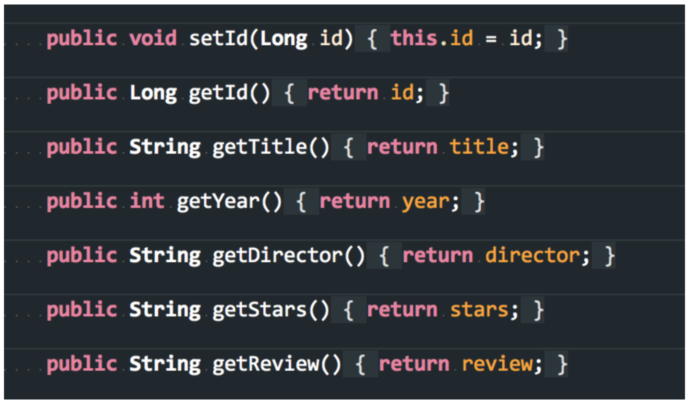
```java
private FilmResource getResource(Film film) {
    return new FilmResource(film, getResourceURI(film.getId()));
}

private Results getResults(Stream<Film> films) {
    final List<FilmResource> filmResources = Lists.newArrayList();

    films.forEach(film -> filmResources.add(getResource(film)));

    return Results.from(filmResources);
}

private Link getResourceURI(Long id) {
    return linkTo(methodOn(this.getClass()).details(id)).withSelfRel();
}

private String decode(String input) {
    try {
        return URLDecoder.decode(input, UTF_8).trim();
    } catch (final UnsupportedEncodingException e) {
        throw new RuntimeException("Unable to decode input", e);
    }
}
```

Figure 13: Film controller utility methods

custom code as possible (in general) is always good as it means you don't have to maintain it yourself and also test it yourself.

4.1 Maven

While Maven (`http://maven.apache.org`) is not technically a library (more a build automation/project management tool), it does come under the umbrella of tools which helped implement a solution. Another well-known tool like this is Gradle (`https://gradle.org`), however this is often used for more complex build management which wasn't really what I needed. I only really needed dependency management within this project as there were a lot of dependencies, and Maven is a lot simpler to set up for this task. Without dependency management I would've had to download individual '.jar' files for each library (which each have their own dependencies to download) and manage them across different computers when developing or testing on other machines. It is also a lot easier to change the version of a dependency if there are compatibility issues between different libraries which rely on each other. Finally as my project was developed in git (`https://git-scm.com`), dependency management meant I did not have to host large jar files in the repository.

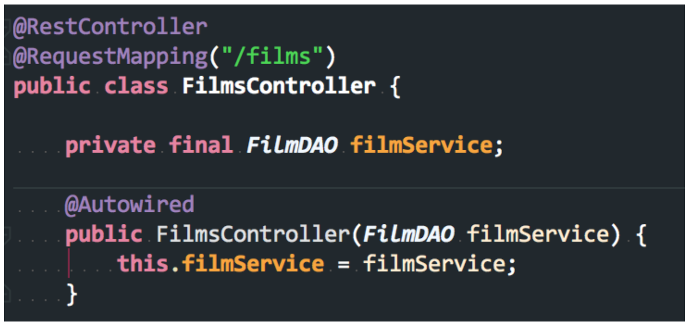


```
    public void setId(Long id) { this.id = id; }

    public Long getId() { return id; }

    public String getTitle() { return title; }

    public int getYear() { return year; }

    public String getDirector() { return director; }

    public String getStars() { return stars; }

    public String getReview() { return review; }
```

Figure 14: Film class methods

4.2 Spring MVC

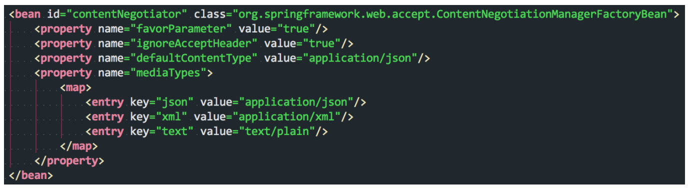
When developing pretty much any Java application, Spring (`https://projects.spring.io/spring-framework/`) is an obvious and hugely popular choice for making developing a lot simpler and more focused on business logic than things like dependency injection. Spring MVC is a part of the framework which focuses on web serving applications traditionally using the MVC structure. Spring MVC allows a REST (Fielding & Taylor 2000) service to be easily configured, which just serves string data to the page without needing to pass it into a JSP. Spring in general allows for much simpler code to be produced as it handles most of the finicky tasks in java that often just waste time. Fulfilling dependencies is the most common (and most useful) part and allows you to focus on more important tasks.



```
@RestController
@RequestMapping("/films")
public class FilmsController {

    private final FilmDAO filmService;

    @Autowired
    public FilmsController(FilmDAO filmService) {
        this.filmService = filmService;
    }
```

Figure 15: Spring's automatic dependency injection

Spring MVC also handles the difficult parts of web service hosting. For example it can handle the response type headers and conversion with no code needed. In the config you can set up which response types you want to allow and how to choose between them and set a default. My config (figure 16) allows a '?format=json' parameter to be put after any url in the servlet, and it will convert to that format no matter which endpoint is being accessed.



```xml
<bean id="contentNegotiator" class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">
    <property name="favorParameter" value="true"/>
    <property name="ignoreAcceptHeader" value="true"/>
    <property name="defaultContentType" value="application/json"/>
    <property name="mediaTypes">
        <map>
            <entry key="json" value="application/json"/>
            <entry key="xml" value="application/xml"/>
            <entry key="text" value="text/plain"/>
        </map>
    </property>
</bean>
```

Figure 16: Response format spring configuration

4.3 Spring HATEOAS

Spring HATEOAS (`http://projects.spring.io/spring-hateoas/`) is a library that extends a lot of Spring MVC functionality to provide REST representations of objects - by following the HATEOAS principles underlined by Fielding & Taylor (2000). This framework allowed me to easily map a resource to its URI and include that in the response body. While I feel that this library is quite cumbersome just for adding a URI to an object, the alternative seemed to require assuming the URI would never change (or require changing it in multiple places if it did). The last alternative was to manually & dynamically find out the servlet URI's, however this seemed needless when Spring HATEOAS does this by design.

4.4 Objectify

Objectify (`https://github.com/objectify/objectify`) is a JPA-like library which allows you to easily connect to Google datastore, and is the library recommended by Google for doing so in a Java program. Previously I had built the DAO using Hibernate (`http://hibernate.org`), however this was not compatible with Google datastore due to non-compliance with Google's proprietary GQL language. While this worked well for what it was required to do, I ended up having to sacrifice some design points that Hibernate had previously allowed me to use. For example, with Hibernate you are able to declare class fields as ‘final’ - with objectify this is not possible as it sets the values after object initialisation.

4.5 Guava

Google Guava (`https://github.com/google/guava`) is a very useful set of libraries that introduce a lot of new functionality to Java. While I originally made more use of this library when I first started the application and have since refactored most uses out of the code, it is still used for array/map initialisation with a static factory, to hide the ugliness of object initialisation that can often be distracting.

4.6 Apache Commons Text

Apache Commons Text (`https://commons.apache.org/proper/commons-text/`) is a set of useful utility libraries dealing with Strings/text. I used this library for its ToStringBuilder class, which uses reflection to create a string representation of a java object - eliminating the need for custom ‘toString’ methods for every data object.

4.7 Gson

Google Gson (`https://github.com/google/gson`) is a library for automatic serialisation from a java object to a JSON string. While Jackson Databind (section 4.8) is able to handle JSON serialisation, I decided to use Gson for this task as Spring MVC has some utilities for integrating with Gson. However I feel as though a relatively similar result could have been achieved using Jackson Databind for both XML and JSON serialisation.

4.8 Jackson Databind

Jackson Databind (`https://github.com/FasterXML/jackson-databind`) is a library for object serialisation, which has several extensions to allow it to convert objects into many different formats such as JSON and XML. Databind is also highly integrated into Spring MVC, so much so that even just having the library in the classpath allows spring to automatically start serialising, without any code needed. This is also possible with JAXB however Databind seems to be the recommended library for Spring MVC so I chose that one instead.

4.9 JQuery

JQuery (`http://jquery.com`) is a front-end JavaScript library which simplifies a wide range of tasks that would be a lot more difficult if written in plain JavaScript. It was very useful for removing a lot of procedural code for doing ajax calls to my web services, from retrieving the data to displaying it on the page. I also used JQuery for handling UI elements such as displaying forms, and laying out object data into an HTML table. I did consider using JQuery UI (`http://jqueryui.com`) however I found it did not offer any UI elements I felt the need to use, so it would have been needless baggage if I had tried to incorporate it into the front-end.

References

Fielding, R. T. & Taylor, R. N. (2000), *Architectural styles and the design of network-based software architectures*, University of California, Irvine Doctoral dissertation.

Krasner, G. E., Pope, S. T. et al. (1988), 'A description of the model-view-controller user interface paradigm in the smalltalk-80 system', *Journal of object oriented programming* **1**(3), 26–49.